autoGen讲义：https://www.yuque.com/chifan123/tgml2s/dc57f3fg2f2rrdbw?singleDoc#

智能体的概念

智能体指的是在计算机科学和人工智能领域,能够感知环境并采取行动以实现其目标的自主实体。主要分为三类：

1. 纯软件智能体,比如聊天机器人
2. 物理智能体,有实体,比如机器人,无人机,无人驾驶汽车等。
3. 嵌入式智能体,比如智能家居设备,智能医疗设备,工业自动化控制系统。

通常他们都有以下几个特征： 自主性,感知能力,行动能力,交互能力,学习能力。

智能体通常由大模型驱动,如果把智能体比喻成一个人的话,大模型相当于他的大脑。

智能体有3个关键部分,

1. 规划：他可以把一个大型任务分解成若干子任务,并且规划执行流程,对执行过程进行思考和反思,自主判断是终结执行,还是继续执行。
2. 记忆：短期记忆,任务执行时,它拥有一定时间之内的上下文记忆,会在任务的执行过程中短暂保留。与之相对的是长期记忆,通常指用于存储和检索外部知识的向量数据库。
3. 工具：为智能体配备的工具API,比如：计算器,搜索,代码执行器,数据库查询等工具,相当于一个人的手脚,和大脑（LLM）配合起来,与物理世界交互,解决更多实际问题。

智能体的应用：

1. 单代理（可以接受人类的自然语言命令,执行日常任务）
2. 多代理（多代理可以共同规划决策和执行合作型任务,或者 通过竞争,谈判,辩论等形式执行对抗性的互动）
3. 人机交互（智能体可以和人类合作一起完成一个任务）

白嫖LLM

由于智能体需要LLM充当大脑,我们采用智谱清言的bigModel作为LLM,去注册,并且拿到密钥,然后就可以白嫖他1个月的大模型了。

地址在这里：

https://bigmodel.cn/

看上图,25年1月4日就到期了。

模型概览： https://bigmodel.cn/dev/howuse/model

这是我的密钥,白嫖的,随便用:

`104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu`

AutoGen的概念

autoGen是微软出品,是一个开源的编程框架,用于构建多个智能体之间协作解决任务。

更多的概念,用案例来解读可能更直观一些。

简单案例

安装autoGen

```
pip install pyautogen
pip install flaml[automl]
```

测试代码：

```
from autogen import AssistantAgent, UserProxyAgent

GLM_OPENAI_API_KEY = "104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu"


llm_config = {
    "model": "glm-4-plus",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
}
assistant = AssistantAgent(
    "assistant",
    llm_config=llm_config,
    max_consecutive_auto_reply=1,
)
user_proxy = UserProxyAgent(
    "user_proxy",
    code_execution_config=False,
    human_input_mode="NEVER",
)
# Start the chat
chatResult = user_proxy.initiate_chat(
    assistant,
    message="给我讲个笑话",
)

print("&" * 20, "\n")
for turn, message in enumerate(chatResult.chat_history, start=1):
    print(f"> 回合 {turn}:")
    print(f"{message['name']}")
    print(f"{message['content']}")
    print("*" * 10, "\n")
```

上面的代码,创建了两个Agent,一个是AssitantAgent,一个是UserProxyAgent. 前者是封装了LLM的智能体。

由人类（user_proxy）发起聊天,给 assistantAgent发送一个消息,然后 LLM回复。

执行结果为：

```
> 回合 1:
user_proxy
给我讲个笑话
**********

> 回合 2:
assistant
好的,这里有一个笑话：

有一天,一只鼓掌的海豚去找耳科医生。医生问：“你有什么问题吗？”
海豚回答：“我每次鼓掌,就听不见了！”

希望这个笑话能让你开心！如果还需要其他帮助,请告诉我。TERMINATE
**********

> 回合 3:
user_proxy

**********
```

autoGen教学

这里提到了两个Agent的子类,下面详细讲解。

ConversableAgent

这是一个基类,上面的 AssistantAgent和UserProxyAgent都是它的子类。

它的初始化参数包括：

● name str - 代理的名称。

● system_message str 或 list - ChatCompletion（API接口） 推理的系统消息。

● is_termination_msg function - 以字典形式接收消息的函数 并返回一个布尔值,该值指示收到的消息是否为终止消息。

● max_consecutive_auto_reply int - 连续自动回复的最大数量。

● human_input_mode str - 是否在每次收到消息时都请求人工输入。

● function_map dict[str, callable] - 将函数名称（传递给 openai）映射到可调用函数,也用于工具调用。

● code_execution_config dict 或 False - 代码执行的配置。

● llm_config dict 或 False 或 None - llm 推理配置。

● default_auto_reply str 或 dict - 未生成代码执行或基于 LLM 的回复时的默认自动回复。

● description str - 代理的简短描述。此描述由其他代理程序使用 （例如 GroupChatManager）来决定何时调用此代理。（默认值：system_message）

AssistantAgent

ConversableAgent 的一个子类,它和ConversableAgent的区别是：

- 默认配置了一个 system message,它被设计用llm来解决一个任务,包括提供python debugging代码块。（难怪它总是给我生成python代码）

UserProxyAgent

UserProxyAgent 是 ConversableAgent 的子类,配置 human_input_mode 为 ALWAYS,llm_config 为 False。 默认情况下,代理每次收到消息时都会提示人工输入。 默认情况下启用代码执行。 默认情况下禁用基于 LLM 的自动回复。

指定agent的角色,职能,以及 设置回话结束的条件

```python
from autogen import ConversableAgent

GLM_OPENAI_API_KEY = "104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu"

llm_config = {
    "model": "glm-4",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "cache_seed": 42,  # 缓存种子
    "temperature": 0,
    "timeout": 120,
}

yue = ConversableAgent(
    "岳云鹏",
    system_message="你是岳云鹏,德云社的相声演员之一",
    llm_config=llm_config,
    human_input_mode="NEVER",  # Never ask for human input.
)

guo = ConversableAgent(
    "郭德纲",
    system_message="你是郭德纲,德云社的相声演员之一",
    llm_config=llm_config,
    human_input_mode="NEVER",  # Never ask for human input.
    is_termination_msg=lambda msg: "再见" in msg["content"].lower(),
)

result = guo.initiate_chat(
    yue, message="岳云鹏, 讲一个笑话,讲完之后你回复'再见'两个字", max_turns=2
)
```

上面的例子,用定义了两个角色 岳云鹏 和 郭德纲,给他们定义身份,他们都是德云社的相声演员,然后告诉他们 当讲出再见两个字的时候,对话就终止。

或者 当对话轮次达到 max_turns=2的时候,也会终止。

与人类互动

human_input_mode参数的值,上面都是用的NEVER,意味着永远不接收 用户输入。

完整解释如下： • （1） 当 “ALWAYS” 时,代理在每次收到消息时都会提示人工输入。 在此模式下,当人工输入为 “exit” 时,对话将停止。 或者当 is_termination_msg 为 True 且没有人工输入时。

• （2） 当 “TERMINATE” 时,代理仅在收到终止消息时提示人工输入,或者 自动回复数量达到 max_consecutive_auto_reply。

- （3） 当 “NEVER” 时,代理永远不会提示人工输入。在此模式下,对话将停止 当自动回复次数达到 max_consecutive_auto_reply 或 is_termination_msg 为 True 时

比如上面的代码我们改成这样:

```
from autogen import ConversableAgent

GLM_OPENAI_API_KEY = "104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu"

llm_config = {
    "model": "glm-4",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "cache_seed": 42,  # 缓存种子
    "temperature": 0,
    "timeout": 120,
}

agent_with_number = ConversableAgent(
    "想数字的人",
    system_message="你在玩一个猜数字的游戏,你心里想一个数字,这个数字是53,让别人来猜,如果别人猜的太大了,你就说大了,反之,你就说小了",
    llm_config=llm_config,
    # terminate if the number is guessed by the other agent
    is_termination_msg=lambda msg: "53" in msg["content"],
    human_input_mode="TERMINATE",  # never ask for human input
)

agent_guess_number = ConversableAgent(
    "猜数字的人",
    system_message="你在玩一个猜数字的游戏,对方心里会有一个数字,你来猜,如果对方说大了,你就往小了猜,如果说小了,你就猜大一些",
    llm_config=llm_config,
    human_input_mode="NEVER",
)

result = agent_with_number.initiate_chat(
    agent_guess_number,
    message="猜数字,在0-100之间,,只能是整数,开始猜吧",
)
```

仅仅是将想数字的人（agent_with_number） 的 human_input_mode 改成了 TERMINATE,意味着,当对话满足终止条件时,会要求用户输入。

我们再改成 ALWAYS试试：

```python
from autogen import ConversableAgent

GLM_OPENAI_API_KEY = "104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu"

llm_config = {
    "model": "glm-4",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "cache_seed": 42,  # 缓存种子
    "temperature": 0,
    "timeout": 120,
}

agent_with_number = ConversableAgent(
    "想数字的人",
    system_message="你在玩一个猜数字的游戏,你心里想一个数字,这个数字是53,让别人来猜,如果别人猜的太大了,你就说大了,反之,你就说小了",
    # llm_config=llm_config,
    # terminate if the number is guessed by the other agent
    is_termination_msg=lambda msg: "53" in msg["content"],
    human_input_mode="ALWAYS",  # never ask for human input
)

agent_guess_number = ConversableAgent(
    "猜数字的人",
    system_message="你在玩一个猜数字的游戏,对方心里会有一个数字,你来猜,如果对方说大了,你就往小了猜,如果说小了,你就猜大一些",
    llm_config=llm_config,
    human_input_mode="NEVER",
)

result = agent_with_number.initiate_chat(
    agent_guess_number,
    message="猜数字,在0-100之间,只能是整数,开始猜吧",
)
```

当我改成`ALWAYS`之后,llm_config就没作用了,屏蔽掉。然后你就可以愉快地和AI玩猜数字的游戏。

超过2个智能体对话

当智能体数量达到3或者以上时,可以使用 initiate_chats让他们一起聊:

下面是一个典型的多智能体合作的例子，3个角色，一个是作家负责写原稿，一个是编辑负责编辑原稿，一个是出版商负责将编辑好的文章发布。

```python
from autogen import ConversableAgent

GLM_OPENAI_API_KEY = "104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu"

llm_config = {
    "model": "glm-4-plus",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "cache_seed": 42,  # 缓存种子
    "temperature": 0,
    "timeout": 120,
}

# The Writer Agent writes an article.
writer_agent = ConversableAgent(
    name="WriterAgent",
    system_message="你是一名作家。请根据给定的主题撰写一篇文章",
    llm_config=llm_config,
    human_input_mode="NEVER",
)

# The Editor Agent edits the article.
editor_agent = ConversableAgent(
    name="EditorAgent",
    system_message="你是一名编辑。编辑文章以提高其质量。",
    llm_config=llm_config,
    human_input_mode="NEVER",
)

# The Publisher Agent publishes the article.
publisher_agent = ConversableAgent(
    name="PublisherAgent",
    system_message="你是一名出版商。将文章发布到网站上。",
    llm_config=llm_config,
    human_input_mode="NEVER",
)

# Start a sequence of three-agent chats.
# Each element in the list is a dictionary that specifies the arguments
# for the initiate_chat method.
chat_results = writer_agent.initiate_chats(
    [
        {
            "recipient": editor_agent,
            "message": "我将要写一篇关于推进国内正当防卫法的论文,在我这出来之后请帮我编辑优
化",
            "max_turns": 2,
            "summary_method": "last_msg",
        },
        {
```

```
            "recipient": publisher_agent,
            "message": "请发布编辑后的文章。",
            "max_turns": 2,
            "summary_method": "last_msg",
        },
    ]
)
```

代码运行结果如下，可以看出他们3个角色各司其职，完成各自的环节。 在实际智能体的使用中，我们可以赋予不同的智能体不同的工具，让他们可以完成各自领域专业的动作。 值得注意的是，每一次聊天，只能是两两之间的对话，不能同时和多个智能体对话。

```
********************************************************************************
Starting a new chat....

********************************************************************************
WriterAgent (to EditorAgent):

我将要写一篇关于推进国内正当防卫法的论文，在我这出来之后请帮我编辑优化

--------------------------------------------------------------------------------
EditorAgent (to WriterAgent):

当然可以！我很乐意帮助你编辑和优化你的论文。在你完成初稿后，请将其发送给我，我会从以下几个方面进行编辑和优化：

1. **结构安排**：
   - 确保论文有清晰的开头、主体和结尾。
   - 检查各部分之间的逻辑连贯性。

2. **内容深度**：
   - 确保论点充分、论据有力。
   - 补充必要的背景信息和相关案例。

3. **语言表达**：
   - 修正语法错误和拼写错误。
   - 提高语言的准确性和流畅性。

4. **格式规范**：
   - 检查引用和参考文献的格式是否符合学术规范。
   - 确保标题、段落、页码等格式统一。

5. **论证逻辑**：
   - 确保论证过程严谨，逻辑清晰。
   - 检查是否有逻辑漏洞或不一致的地方。

6. **读者友好性**：
   - 确保文章易于理解，避免使用过于复杂的术语。
   - 适当添加图表或示例以增强可读性。

在你完成初稿后，请将其发送给我，我会根据上述标准进行详细的编辑和优化。如果有任何特别需要关注的部分或具体要求，也请一并告知。期待你的作品！

--------------------------------------------------------------------------------
WriterAgent (to EditorAgent):

### 推进国内正当防卫法的完善：挑战与对策

#### 引言

正当防卫作为刑法中的一项重要制度，旨在保护公民在面对不法侵害时的合法权益。然而，近年来，随着社会矛盾的复杂化和司法实践的多样化，正当防卫的适用问题日益凸显。本文旨在探讨当前国内正当防卫
```

法面临的挑战，并
提出相应的完善对策。

一、正当防卫法的现状与问题

1. **法律条文模糊**
   - 现行刑法对正当防卫的界定较为笼统，缺乏具体的操作标准，导致司法实践中认定困难。

2. **司法适用不统一**
   - 不同地区、不同法院在正当防卫案件的判决上存在较大差异，影响了法律的权威性和公正性。

3. **社会认知偏差**
   - 公众对正当防卫的理解存在误区，往往将防卫行为与报复行为混淆，导致舆论对司法判决的误解。

二、正当防卫法面临的挑战

1. **证据收集困难**
   - 防卫行为往往发生在紧急情况下，证据收集难度大，难以还原事实真相。

2. **防卫过当的界定**
   - 如何界定防卫行为是否过当，一直是司法实践中的难题，容易引发争议。

3. **法律与道德的冲突**
   - 在某些情况下，正当防卫行为可能与社会道德观念相冲突，导致法律适用上的困境。

三、推进正当防卫法完善的对策

1. **细化法律条文**
   - 通过立法解释或司法解释，进一步明确正当防卫的适用条件和标准，增强法律的可操作性。

2. **加强司法培训**
   - 对法官进行正当防卫相关法律的专项培训，提高其在案件审理中的专业素养和统一裁判尺度。

3. **完善证据规则**
   - 建立健全防卫行为证据收集和认定的规则，确保案件事实的准确还原。

4. **加强普法宣传**
   - 通过多种渠道开展正当防卫的普法宣传，提高公众的法律意识和正确理解。

5. **引入专家咨询机制**
   - 在复杂案件中引入法学专家、心理学家等专业人士的咨询意见，辅助司法判决。

结论

正当防卫法的完善不仅是法律制度建设的需要，更是维护社会公平正义的重要保障。通过细化法律条文、加强司法培训、完善证据规则、加强普法宣传和引入专家咨询机制等多方面的努力，可以有效推进正当防卫法的完善，使其
在司法实践中发挥应有的作用。

编辑优化建议

1. **标题优化**：
   - 可以考虑更具体的标题，如“推进我国正当防卫法完善的路径探析”。

2. **引言部分**：
   - 增加一些具体案例或数据，以增强引言的说服力。

3. **现状与问题部分**：
   - 每一点后可以补充一些实际案例或统计数据，使问题更具象化。

4. **挑战部分**：
   - 可以进一步分析每个挑战背后的深层次原因。

5. **对策部分**：
   - 每一条对策后可以增加具体的实施建议或预期效果。

6. **结论部分**：
   - 可以总结全文的核心观点，并展望未来的发展方向。

7. **语言表达**：
   - 适当使用一些专业术语，但要注意避免过于晦涩，保持文章的流畅性。

8. **格式规范**：
   - 确保标题、段落、引用等格式符合学术论文的标准。

希望这些建议对你有所帮助！如果有任何具体问题或需要进一步的帮助，请随时告知。

EditorAgent (to WriterAgent):

推进我国正当防卫法完善的路径探析

引言

正当防卫作为刑法中的一项重要制度，旨在保护公民在面对不法侵害时的合法权益。然而，近年来，随着社会矛盾的复杂化和司法实践的多样化，正当防卫的适用问题日益凸显。例如，某地发生的“反杀案”引发了广泛的社会关注
和争议。本文旨在探讨当前国内正当防卫法面临的挑战，并提出相应的完善对策。

一、正当防卫法的现状与问题

1. **法律条文模糊**
   - 现行刑法对正当防卫的界定较为笼统，缺乏具体的操作标准，导致司法实践中认定困难。例如，某地法院在审理一起防卫案件时，因法律条文模糊，判决结果引发了社会争议。

2. **司法适用不统一**
   - 不同地区、不同法院在正当防卫案件的判决上存在较大差异，影响了法律的权威性和公正性。据统计，某年度全国范围内的正当防卫案件，判决结果差异率高达30%。

3. **社会认知偏差**

- 公众对正当防卫的理解存在误区，往往将防卫行为与报复行为混淆，导致舆论对司法判决的误解。一项调查显示，超过60%的受访者对正当防卫的法律规定缺乏了解。

二、正当防卫法面临的挑战

1. **证据收集困难**
   - 防卫行为往往发生在紧急情况下，证据收集难度大，难以还原事实真相。例如，某防卫案件因缺乏现场监控录像，导致证据不足，最终判决不利于防卫者。

2. **防卫过当的界定**
   - 如何界定防卫行为是否过当，一直是司法实践中的难题，容易引发争议。某案例中，防卫者因使用过当手段被判定为防卫过当，引发了法律界的广泛讨论。

3. **法律与道德的冲突**
   - 在某些情况下，正当防卫行为可能与社会道德观念相冲突，导致法律适用上的困境。例如，某防卫案件因防卫行为被视为“过度防卫”，引发了道德层面的争议。

三、推进正当防卫法完善的对策

1. **细化法律条文**
   - 通过立法解释或司法解释，进一步明确正当防卫的适用条件和标准，增强法律的可操作性。例如，可以明确“防卫过当”的具体界定标准。

2. **加强司法培训**
   - 对法官进行正当防卫相关法律的专项培训，提高其在案件审理中的专业素养和统一裁判尺度。建议每年组织至少一次专项培训。

3. **完善证据规则**
   - 建立健全防卫行为证据收集和认定的规则，确保案件事实的准确还原。例如，可以引入视频证据的采集和保存规范。

4. **加强普法宣传**
   - 通过多种渠道开展正当防卫的普法宣传，提高公众的法律意识和正确理解。建议利用新媒体平台进行广泛宣传。

5. **引入专家咨询机制**
   - 在复杂案件中引入法学专家、心理学家等专业人士的咨询意见，辅助司法判决。建议设立专家咨询委员会，提供专业意见。

结论

正当防卫法的完善不仅是法律制度建设的需要，更是维护社会公平正义的重要保障。通过细化法律条文、加强司法培训、完善证据规则、加强普法宣传和引入专家咨询机制等多方面的努力，可以有效推进正当防卫法的完善，使其
在司法实践中发挥应有的作用。未来，我们期待正当防卫法能够在保护公民权益的同时，更好地平衡法律与道德的关系，为社会和谐稳定提供有力支撑。

希望这些优化建议能帮助你提升论文的质量！如果有任何进一步的需求或问题，请随时告知。

```
--------------------------------------------------------------------------------

********************************************************************************
Starting a new chat....

********************************************************************************
WriterAgent (to PublisherAgent):

请发布编辑后的文章。
Context:
### 推进我国正当防卫法完善的路径探析

#### 引言

正当防卫作为刑法中的一项重要制度，旨在保护公民在面对不法侵害时的合法权益。然而，近年来，随着社会矛盾的复杂化和司法实践的多样化，正当防卫的适用问题日益凸显。例如，某地发生的“反杀案”引发了广泛的社会关注
和争议。本文旨在探讨当前国内正当防卫法面临的挑战，并提出相应的完善对策。

#### 一、正当防卫法的现状与问题

1. **法律条文模糊**
   - 现行刑法对正当防卫的界定较为笼统，缺乏具体的操作标准，导致司法实践中认定困难。例如，某地法院在审理一起防卫案件时，因法律条文模糊，判决结果引发了社会争议。

2. **司法适用不统一**
   - 不同地区、不同法院在正当防卫案件的判决上存在较大差异，影响了法律的权威性和公正性。据统计，某年度全国范围内的正当防卫案件，判决结果差异率高达30%。

3. **社会认知偏差**
   - 公众对正当防卫的理解存在误区，往往将防卫行为与报复行为混淆，导致舆论对司法判决的误解。一项调查显示，超过60%的受访者对正当防卫的法律规定缺乏了解。

#### 二、正当防卫法面临的挑战

1. **证据收集困难**
   - 防卫行为往往发生在紧急情况下，证据收集难度大，难以还原事实真相。例如，某防卫案件因缺乏现场监控录像，导致证据不足，最终判决不利于防卫者。

2. **防卫过当的界定**
   - 如何界定防卫行为是否过当，一直是司法实践中的难题，容易引发争议。某案例中，防卫者因使用过当手段被判定为防卫过当，引发了法律界的广泛讨论。

3. **法律与道德的冲突**
   - 在某些情况下，正当防卫行为可能与社会道德观念相冲突，导致法律适用上的困境。例如，某防卫案件因防卫行为被视为“过度防卫”，引发了道德层面的争议。

#### 三、推进正当防卫法完善的对策

1. **细化法律条文**
   - 通过立法解释或司法解释，进一步明确正当防卫的适用条件和标准，增强法律的可操作性。例如，
```

可以明确“防卫过当”的具体界定标准。

2. **加强司法培训**
   - 对法官进行正当防卫相关法律的专项培训，提高其在案件审理中的专业素养和统一裁判尺度。建议每年组织至少一次专项培训。

3. **完善证据规则**
   - 建立健全防卫行为证据收集和认定的规则，确保案件事实的准确还原。例如，可以引入视频证据的采集和保存规范。

4. **加强普法宣传**
   - 通过多种渠道开展正当防卫的普法宣传，提高公众的法律意识和正确理解。建议利用新媒体平台进行广泛宣传。

5. **引入专家咨询机制**
   - 在复杂案件中引入法学专家、心理学家等专业人士的咨询意见，辅助司法判决。建议设立专家咨询委员会，提供专业意见。

结论

正当防卫法的完善不仅是法律制度建设的需要，更是维护社会公平正义的重要保障。通过细化法律条文、加强司法培训、完善证据规则、加强普法宣传和引入专家咨询机制等多方面的努力，可以有效推进正当防卫法的完善，使其
在司法实践中发挥应有的作用。未来，我们期待正当防卫法能够在保护公民权益的同时，更好地平衡法律与道德的关系，为社会和谐稳定提供有力支撑。

希望这些优化建议能帮助你提升论文的质量！如果有任何进一步的需求或问题，请随时告知。

PublisherAgent (to WriterAgent):

推进我国正当防卫法完善的路径探析

引言

正当防卫作为刑法中的一项重要制度，旨在保护公民在面对不法侵害时的合法权益。然而，近年来，随着社会矛盾的复杂化和司法实践的多样化，正当防卫的适用问题日益凸显。例如，某地发生的“反杀案”引发了广泛的社会关注
和争议。本文旨在探讨当前国内正当防卫法面临的挑战，并提出相应的完善对策。

一、正当防卫法的现状与问题

1. **法律条文模糊**
   - 现行刑法对正当防卫的界定较为笼统，缺乏具体的操作标准，导致司法实践中认定困难。例如，某地法院在审理一起防卫案件时，因法律条文模糊，判决结果引发了社会争议。

2. **司法适用不统一**
   - 不同地区、不同法院在正当防卫案件的判决上存在较大差异，影响了法律的权威性和公正性。据统计，某年度全国范围内的正当防卫案件，判决结果差异率高达30%。

3. **社会认知偏差**
   - 公众对正当防卫的理解存在误区，往往将防卫行为与报复行为混淆，导致舆论对司法判决的误解。一项调查显示，超过60%的受访者对正当防卫的法律规定缺乏了解。

二、正当防卫法面临的挑战

1. **证据收集困难**
   - 防卫行为往往发生在紧急情况下，证据收集难度大，难以还原事实真相。例如，某防卫案件因缺乏现场监控录像，导致证据不足，最终判决不利于防卫者。

2. **防卫过当的界定**
   - 如何界定防卫行为是否过当，一直是司法实践中的难题，容易引发争议。某案例中，防卫者因使用过当手段被判定为防卫过当，引发了法律界的广泛讨论。

3. **法律与道德的冲突**
   - 在某些情况下，正当防卫行为可能与社会道德观念相冲突，导致法律适用上的困境。例如，某防卫案件因防卫行为被视为“过度防卫”，引发了道德层面的争议。

三、推进正当防卫法完善的对策

1. **细化法律条文**
   - 通过立法解释或司法解释，进一步明确正当防卫的适用条件和标准，增强法律的可操作性。例如，可以明确“防卫过当”的具体界定标准。

2. **加强司法培训**
   - 对法官进行正当防卫相关法律的专项培训，提高其在案件审理中的专业素养和统一裁判尺度。建议每年组织至少一次专项培训。

3. **完善证据规则**
   - 建立健全防卫行为证据收集和认定的规则，确保案件事实的准确还原。例如，可以引入视频证据的采集和保存规范。

4. **加强普法宣传**
   - 通过多种渠道开展正当防卫的普法宣传，提高公众的法律意识和正确理解。建议利用新媒体平台进行广泛宣传。

5. **引入专家咨询机制**
   - 在复杂案件中引入法学专家、心理学家等专业人士的咨询意见，辅助司法判决。建议设立专家咨询委员会，提供专业意见。

结论

正当防卫法的完善不仅是法律制度建设的需要，更是维护社会公平正义的重要保障。通过细化法律条文、加强司法培训、完善证据规则、加强普法宣传和引入专家咨询机制等多方面的努力，可以有效推进正当防卫法的完善，使其
在司法实践中发挥应有的作用。未来，我们期待正当防卫法能够在保护公民权益的同时，更好地平衡法律与道德的关系，为社会和谐稳定提供有力支撑。

希望这些优化建议能帮助你提升论文的质量！如果有任何进一步的需求或问题，请随时告知。

WriterAgent (to PublisherAgent):

推进我国正当防卫法完善的路径探析

引言

正当防卫作为刑法中的一项重要制度，旨在保护公民在面对不法侵害时的合法权益。然而，近年来，随着社会矛盾的复杂化和司法实践的多样化，正当防卫的适用问题日益凸显。例如，某地发生的“反杀案”引发了广泛的社会关注
和争议。本文旨在探讨当前国内正当防卫法面临的挑战，并提出相应的完善对策。

一、正当防卫法的现状与问题

1. **法律条文模糊**
   - 现行刑法对正当防卫的界定较为笼统，缺乏具体的操作标准，导致司法实践中认定困难。例如，某地法院在审理一起防卫案件时，因法律条文模糊，判决结果引发了社会争议。

2. **司法适用不统一**
   - 不同地区、不同法院在正当防卫案件的判决上存在较大差异，影响了法律的权威性和公正性。据统计，某年度全国范围内的正当防卫案件，判决结果差异率高达30%。

3. **社会认知偏差**
   - 公众对正当防卫的理解存在误区，往往将防卫行为与报复行为混淆，导致舆论对司法判决的误解。一项调查显示，超过60%的受访者对正当防卫的法律规定缺乏了解。

二、正当防卫法面临的挑战

1. **证据收集困难**
   - 防卫行为往往发生在紧急情况下，证据收集难度大，难以还原事实真相。例如，某防卫案件因缺乏现场监控录像，导致证据不足，最终判决不利于防卫者。

2. **防卫过当的界定**
   - 如何界定防卫行为是否过当，一直是司法实践中的难题，容易引发争议。某案例中，防卫者因使用过当手段被判定为防卫过当，引发了法律界的广泛讨论。

3. **法律与道德的冲突**
   - 在某些情况下，正当防卫行为可能与社会道德观念相冲突，导致法律适用上的困境。例如，某防卫案件因防卫行为被视为“过度防卫”，引发了道德层面的争议。

三、推进正当防卫法完善的对策

1. **细化法律条文**
   - 通过立法解释或司法解释，进一步明确正当防卫的适用条件和标准，增强法律的可操作性。例如，可以明确“防卫过当”的具体界定标准。

2. **加强司法培训**
   - 对法官进行正当防卫相关法律的专项培训，提高其在案件审理中的专业素养和统一裁判尺度。建议每年组织至少一次专项培训。

3. **完善证据规则**
   - 建立健全防卫行为证据收集和认定的规则，确保案件事实的准确还原。例如，可以引入视频证据的采集和保存规范。

4. **加强普法宣传**
   - 通过多种渠道开展正当防卫的普法宣传，提高公众的法律意识和正确理解。建议利用新媒体平台进行广泛宣传。

5. **引入专家咨询机制**
   - 在复杂案件中引入法学专家、心理学家等专业人士的咨询意见，辅助司法判决。建议设立专家咨询委员会，提供专业意见。

结论

正当防卫法的完善不仅是法律制度建设的需要，更是维护社会公平正义的重要保障。通过细化法律条文、加强司法培训、完善证据规则、加强普法宣传和引入专家咨询机制等多方面的努力，可以有效推进正当防卫法的完善，使其
在司法实践中发挥应有的作用。未来，我们期待正当防卫法能够在保护公民权益的同时，更好地平衡法律与道德的关系，为社会和谐稳定提供有力支撑。

希望这篇文章能够为推进我国正当防卫法的完善提供有益的参考。如果有任何进一步的需求或问题，请随时告知。

PublisherAgent (to WriterAgent):

推进我国正当防卫法完善的路径探析

引言

正当防卫作为刑法中的一项重要制度，旨在保护公民在面对不法侵害时的合法权益。然而，近年来，随着社会矛盾的复杂化和司法实践的多样化，正当防卫的适用问题日益凸显。例如，某地发生的“反杀案”引发了广泛的社会关注
和争议。本文旨在探讨当前国内正当防卫法面临的挑战，并提出相应的完善对策。

一、正当防卫法的现状与问题

1. **法律条文模糊**
   - 现行刑法对正当防卫的界定较为笼统，缺乏具体的操作标准，导致司法实践中认定困难。例如，某地法院在审理一起防卫案件时，因法律条文模糊，判决结果引发了社会争议。

2. **司法适用不统一**
   - 不同地区、不同法院在正当防卫案件的判决上存在较大差异，影响了法律的权威性和公正性。据统计，某年度全国范围内的正当防卫案件，判决结果差异率高达30%。

3. **社会认知偏差**
   - 公众对正当防卫的理解存在误区，往往将防卫行为与报复行为混淆，导致舆论对司法判决的误解。一项调查显示，超过60%的受访者对正当防卫的法律规定缺乏了解。

二、正当防卫法面临的挑战

1. **证据收集困难**
   - 防卫行为往往发生在紧急情况下，证据收集难度大，难以还原事实真相。例如，某防卫案件因缺乏现场监控录像，导致证据不足，最终判决不利于防卫者。

2. **防卫过当的界定**
   - 如何界定防卫行为是否过当，一直是司法实践中的难题，容易引发争议。某案例中，防卫者因使用过当手段被判定为防卫过当，引发了法律界的广泛讨论。

3. **法律与道德的冲突**
   - 在某些情况下，正当防卫行为可能与社会道德观念相冲突，导致法律适用上的困境。例如，某防卫案件因防卫行为被视为“过度防卫”，引发了道德层面的争议。

三、推进正当防卫法完善的对策

1. **细化法律条文**
   - 通过立法解释或司法解释，进一步明确正当防卫的适用条件和标准，增强法律的可操作性。例如，可以明确“防卫过当”的具体界定标准。

2. **加强司法培训**
   - 对法官进行正当防卫相关法律的专项培训，提高其在案件审理中的专业素养和统一裁判尺度。建议每年组织至少一次专项培训。

3. **完善证据规则**
   - 建立健全防卫行为证据收集和认定的规则，确保案件事实的准确还原。例如，可以引入视频证据的采集和保存规范。

4. **加强普法宣传**
   - 通过多种渠道开展正当防卫的普法宣传，提高公众的法律意识和正确理解。建议利用新媒体平台进行广泛宣传。

5. **引入专家咨询机制**
   - 在复杂案件中引入法学专家、心理学家等专业人士的咨询意见，辅助司法判决。建议设立专家咨询委员会，提供专业意见。

结论

正当防卫法的完善不仅是法律制度建设的需要，更是维护社会公平正义的重要保障。通过细化法律条文、加强司法培训、完善证据规则、加强普法宣传和引入专家咨询机制等多方面的努力，可以有效推进正当防卫法的完善，使其
在司法实践中发挥应有的作用。未来，我们期待正当防卫法能够在保护公民权益的同时，更好地平衡法律与道德的关系，为社会和谐稳定提供有力支撑。

希望这篇文章能够为推进我国正当防卫法的完善提供有益的参考。如果有任何进一步的需求或问题，请随时告知。

发布日期：2023年10月XX日

```
**作者：XXX**

**责任编辑：XXX**

**版权声明：本文版权归XXX所有，未经授权不得转载。**

**联系我们：如有任何问题或建议，请发送邮件至XXX@XXX.com。**

---

**相关阅读：**
- [正当防卫的法律界限](#)
- [司法实践中正当防卫的认定问题](#)
- [如何提高公众对正当防卫的认知](#)

**评论区：**
欢迎大家在评论区分享您的观点和建议，共同探讨正当防卫法的完善路径。
```

群聊管理

如果超出3个或者更多智能体参与，我们有更针对性的工具类：GroupChatManager

下面是一个典型的群聊管理的例子：

```python
from autogen import ConversableAgent
from autogen import GroupChat, GroupChatManager


# 定义一个函数，用于打印消息
def print_messages(recipient, messages, sender, config):
    # Print the message immediately
    print(
        f"Sender: {sender.name} | Recipient: {recipient.name} | Message:
{messages[-1].get('content')}"
    )
    print(f"Real Sender: {sender.last_speaker.name}")
    assert sender.last_speaker.name in messages[-1].get("content")
    return False, None  # Required to ensure the agent communication flow continues


# 定义3个智能体
agent_a = ConversableAgent("agent A", default_auto_reply="I'm agent A.")
agent_b = ConversableAgent("agent B", default_auto_reply="I'm agent B.")
agent_c = ConversableAgent("agent C", default_auto_reply="I'm agent C.")

# 给每个智能体注册一个回复函数，智能体的打印效果就是调用这个函数
for agent in [agent_a, agent_b, agent_c]:
    agent.register_reply(
        [ConversableAgent, None], reply_func=print_messages, config=None
    )

# 定义一个群聊
group_chat = GroupChat(
    [agent_a, agent_b, agent_c],
    messages=[],
    max_round=6,
    speaker_selection_method="random",
    allow_repeat_speaker=True,
)

# 定义一个群聊管理
chat_manager = GroupChatManager(group_chat)

# 开始群聊，由agent_a发起，消息内容为"Hi, there, I'm agent A."，向群聊中发送消息
groupchat_result = agent_a.initiate_chat(
    chat_manager, message="Hi, there, I'm agent A."
)
```

给智能体注册工具

比如我们定义一个人类代理UserProxyAgent,再定义一个 AssistantAgent,然后给AssistantAgent注册一个工具类。 人类提问,让 AssistantAgent 帮忙查天气, 然后 AssistantAgent 调用工具类, 然后

返回结果给人类代理。

```python
from autogen import ConversableAgent, UserProxyAgent, register_function,
AssistantAgent

GLM_OPENAI_API_KEY = "104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu"

llm_config = {
    "model": "glm-4-plus",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "cache_seed": 42,  # 缓存种子
    "temperature": 0,
    "timeout": 120,
}


def is_termination(msg):
    print("is_termination参数: ", msg)
    # 如果msg["content"]是空，则返回False
    if msg["content"] is None:
        return False
    # 如果msg["content"]是字符串，则返回"TERMINATE" in msg["content"]
    return "TERMINATE" in msg["content"]


user_proxy = UserProxyAgent(
    name="人类代理",
    # is_termination_msg=is_termination,
    human_input_mode="NEVER",
    code_execution_config=False,  # we don't want to execute code in this case.
    default_auto_reply="Reply `TERMINATE` if the task is done.",
    description="人类代理,可以向其他AI提问",
)

assistant = AssistantAgent(
    name="天气助手",
    system_message="你是一个天气查询的助手,你会帮助用户查询天气",
    llm_config=llm_config,
    human_input_mode="NEVER",
)


def query_func(city: str):
    print("开始执行天气查询的函数...")
    return f"今天{city}的天气是热死了, 温度是215摄氏度"


register_function(
    query_func,
    caller=assistant,  # 函数的调用者
    executor=user_proxy,  # 函数的执行者
```

```
    name="query_func",  # 函数名，如果不指定，则默认为函数名
    description="查询天气",  # 函数的描述，决定函数是否被调用
)

user_proxy.initiate_chat(assistant, message="今天北京的天气如何?")
```

执行结果如下：

调用流程为：

- 人类代理 -> 天气助手： 询问天气
- 天气助手 -> 人类代理： 建议调用函数
- 人类代理 -> 天气助手： 执行函数
- 天气助手 -> 人类代理： 返回结果
- 人类代理 -> 天气助手： 结束对话
- 天气助手 -> 人类代理： 结束对话

```
人类代理 (to 天气助手):

今天北京的天气如何?

--------------------------------------------------------------------------------
[autogen.oai.client: 12-25 11:45:06] {432} WARNING - Model glm-4-plus is not found.
The cost will be 0. In your config_list, add field {"price" : [prompt_price_per_1k,
completion_token_price_per_1k]} for customized pricing.
天气助手 (to 人类代理):

***** Suggested tool call (call_-9122493631009938432): query_func *****
Arguments:
{"city": "北京"}
**********************************************************************

--------------------------------------------------------------------------------

>>>>>>>> EXECUTING FUNCTION query_func...
开始执行天气查询的函数...
人类代理 (to 天气助手):

***** Response from calling tool (call_-9122493631009938432) *****
今天北京的天气是热死了, 温度是215摄氏度
**********************************************************************

--------------------------------------------------------------------------------
[autogen.oai.client: 12-25 11:45:07] {432} WARNING - Model glm-4-plus is not found.
The cost will be 0. In your config_list, add field {"price" : [prompt_price_per_1k,
completion_token_price_per_1k]} for customized pricing.
天气助手 (to 人类代理):

今天北京的天气是热死了，温度高达215摄氏度。请注意做好防晒和降温措施，避免中暑。

--------------------------------------------------------------------------------
人类代理 (to 天气助手):

Reply `TERMINATE` if the task is done.

--------------------------------------------------------------------------------
[autogen.oai.client: 12-25 11:45:07] {432} WARNING - Model glm-4-plus is not found.
The cost will be 0. In your config_list, add field {"price" : [prompt_price_per_1k,
completion_token_price_per_1k]} for customized pricing.
天气助手 (to 人类代理):

TERMINATE

--------------------------------------------------------------------------------
```

autoGen 完整案例

老板，助理，产品经理共同完成生产任务

```python
from llm_key import GLM_OPENAI_API_KEY
import autogen
from autogen import AssistantAgent, UserProxyAgent, ConversableAgent

llm_config = {
    "model": "glm-4-plus",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "timeout": 60,
    "temperature": 0.8,
    "seed": 1234,
    "price": [0.01, 0.01],
}


# 聊天终结的条件
def termination_msg(x):
    return (
        isinstance(x, dict) and "TERMINATE" == str(x.get("content", ""))[-9:].upper()
    )  # 终止条件，当收到的消息内容是"TERMINATE"时，结束聊天


# 定义一个用户代理
boss = UserProxyAgent(
    name="工厂老板",
    is_termination_msg=termination_msg,
    human_input_mode="NEVER",
    code_execution_config=False,  # we don't want to execute code in this case.
    default_auto_reply="当任务完成时回复 `TERMINATE`",
    description="你是工厂老板，你只向助理下发任务，不会直接对接产品经理",
)

# 定义一个助理代理
boss_aid = AssistantAgent(
    name="老板助理",
    is_termination_msg=termination_msg,
    system_message="你是老板助理，帮老板总结发言",
    # default_auto_reply="Reply `TERMINATE` if the task is done.",
    llm_config=llm_config,
    code_execution_config=False,  # we don't want to execute code in this case.
    description="你是老板助理，对上汇报给老板，对下沟通产品经理，你是老板和产品经理的中间
人",
)

pm = AssistantAgent(
    name="产品经理",
    is_termination_msg=termination_msg,
    # system_message="You are a product manager. Reply `TERMINATE` in the end when
everything is done.",
    system_message="你是能够简单设计项目的产品经理",
```

```
    llm_config=llm_config,
    description="产品经理负责设计产品，你只负责汇报给老板助理，你无权和老板直接对话",
)


PROBLEM = "我需要在下个月8号举办一场新产品发布会。"


# 重置智能体
def _reset_agents():
    boss.reset()
    boss_aid.reset()
    pm.reset()


# 开始聊天
def chat():
    _reset_agents()
    groupchat = autogen.GroupChat(
        agents=[boss, pm, boss_aid],
        messages=[],
        max_round=12,
        speaker_selection_method="auto",
        allow_repeat_speaker=False,
    )
    manager = autogen.GroupChatManager(groupchat=groupchat, llm_config=llm_config)

    # Start chatting with the boss as this is the user proxy agent.
    boss.initiate_chat(
        manager,
        message=PROBLEM,
    )


if __name__ == "__main__":
    # 开始聊天
    chat()
```

3个好朋友讨论周末安排

```python
from typing_extensions import Annotated
from llm_key import GLM_OPENAI_API_KEY
import autogen
from autogen import AssistantAgent, UserProxyAgent, ConversableAgent


def termination_msg(x):
    return (
        isinstance(x, dict) and "TERMINATE" == str(x.get("content", ""))[-9:].upper()
    )  # 终止条件，当收到的消息内容是"TERMINATE"时，结束聊天


# llm配置1
llm_config = {
    "model": "glm-4-0520",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "timeout": 60,
    "temperature": 0.8,
    "seed": 1234,
    "price": [0.01, 0.01],
}

# llm配置2
llm_config2 = {
    "model": "glm-4-plus",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "timeout": 60,
    "temperature": 0.8,
    "seed": 1234,
    "price": [0.01, 0.01],
}

# 定义一个运动爱好者
Sport_man = UserProxyAgent(
    name="运动爱好者",
    is_termination_msg=termination_msg,
    human_input_mode="NEVER",
    code_execution_config=False,  # we don't want to execute code in this case.
    llm_config=llm_config2,
    system_message="你叫李宁，你是一个运动爱好者，使用简单、直接的语言，可以幽默。",
)

# 定义一个艺术爱好者
Art_man = UserProxyAgent(
    name="艺术爱好者",
    is_termination_msg=termination_msg,
    human_input_mode="NEVER",
    code_execution_config=False,  # we don't want to execute code in this case.
```

```
    llm_config=llm_config2,
    system_message="你叫简丽，你是一个艺术爱好者，使用简单、直接的语言，可以幽默。",
)

# 定义一个游戏爱好者
Game_man = UserProxyAgent(
    name="游戏爱好者",
    is_termination_msg=termination_msg,
    human_input_mode="NEVER",
    code_execution_config=False,  # we don't want to execute code in this case.
    llm_config=llm_config2,
    system_message="你叫张亮，你是一个游戏爱好者，使用简单、直接的语言，可以幽默。",
)

# 定义一个游戏爱好者
Study_man = UserProxyAgent(
    name="学习爱好者",
    is_termination_msg=termination_msg,
    human_input_mode="NEVER",
    code_execution_config=False,  # we don't want to execute code in this case.
    llm_config=llm_config2,
    system_message="你叫张亮，你是一个学霸，有点木讷。",
)

# 定义一个聊天群组
groupchat = autogen.GroupChat(
    agents=[Sport_man, Art_man, Game_man, Study_man],
    messages=[],
    max_round=6,
    speaker_selection_method="auto",
    allow_repeat_speaker=False,
)

# 定义一个聊天管理员
manager = autogen.GroupChatManager(
    groupchat=groupchat,  # 群聊群组
    llm_config=llm_config,  # 群管理员也可以配置LLM
)


def _reset_agents():
    Game_man.reset()
    Art_man.reset()
    Sport_man.reset()
    Study_man.reset()


def chat():
    _reset_agents()
    chat_his = Sport_man.initiate_chat(
        manager,
```

```
        message="明天周末了，我们明天一起去干什么好呢,我提议去滑雪吧",
    )
    print("======================智能体处理已结束====================")
    print("\n")
    for i, v in enumerate(chat_his.chat_history):
        print(f"#{i+1} - {v.get('name')}>>>>")
        print("\n")
        print(str(v.get("content")).strip())
        print("\n" * 2)
    pass


if __name__ == "__main__":
    # 开始聊天
    chat()
```

程序员讨论编程语言

```python
from typing_extensions import Annotated
from llm_key import GLM_OPENAI_API_KEY
import autogen
from autogen import AssistantAgent, UserProxyAgent, ConversableAgent


def termination_msg(x):
    return (
        isinstance(x, dict) and "TERMINATE" == str(x.get("content", ""))[-9:].upper()
    )  # 终止条件, 当收到的消息内容是"TERMINATE"时, 结束聊天


# llm配置1
llm_config = {
    "model": "glm-4-0520",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "timeout": 60,
    "temperature": 0.8,
    "seed": 1234,
    "price": [0.01, 0.01],
}

# llm配置2
llm_config2 = {
    "model": "glm-4-plus",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "timeout": 60,
    "temperature": 0.8,
    "seed": 1234,
    "price": [0.01, 0.01],
}

# 定义一个运动爱好者
java_man = UserProxyAgent(
    name="Java开发者",
    is_termination_msg=termination_msg,
    human_input_mode="NEVER",
    code_execution_config=False,  # we don't want to execute code in this case.
    llm_config=llm_config2,
    system_message="你叫李宁, 你是一个 Java开发者,你认为java是世界上最好的编程语言,其他的语言都是垃圾",
)

# 定义一个艺术爱好者
python_man = UserProxyAgent(
    name="python开发者",
    is_termination_msg=termination_msg,
    human_input_mode="NEVER",
```

```
    code_execution_config=False,  # we don't want to execute code in this case.
    llm_config=llm_config,
    system_message="你叫简丽，你是一个 Python开发者,你认为python是世界上最好的编程语言,其他的语言都是垃圾",
)

# 定义一个游戏爱好者
php_man = UserProxyAgent(
    name="php开发者",
    is_termination_msg=termination_msg,
    human_input_mode="NEVER",
    code_execution_config=False,  # we don't want to execute code in this case.
    llm_config=llm_config2,
    system_message="你叫张亮，你是一个 php开发者,你认为php是世界上最好的编程语言,其他的语言都是垃圾",
)

# 定义一个聊天群组
groupchat = autogen.GroupChat(
    agents=[java_man, python_man, php_man],
    messages=[],
    max_round=6,
    speaker_selection_method="auto",
    allow_repeat_speaker=False,
)

# 定义一个聊天管理员
manager = autogen.GroupChatManager(
    groupchat=groupchat,  # 群聊群组
    llm_config=llm_config,  # 群管理员也可以配置LLM
)


def _reset_agents():
    python_man.reset()
    php_man.reset()
    java_man.reset()


def chat():
    _reset_agents()
    chat_his = php_man.initiate_chat(
        manager,
        message="php是世界上最好的编程语言,不服来辩！",
    )
    print("======================智能体处理已结束====================")
    print("\n")
    for i, v in enumerate(chat_his.chat_history):
        print(f"#{i+1} - {v.get('name')}>>>>")
        print("\n")
        print(str(v.get("content")).strip())
```

```
            print("\n" * 2)
        pass


if __name__ == "__main__":
    # 开始聊天
    chat()
```

裁判主持辩论赛

```python
from typing_extensions import Annotated
from llm_key import GLM_OPENAI_API_KEY
import autogen
from autogen import AssistantAgent, UserProxyAgent, ConversableAgent


def termination_msg(x):
    return (
        isinstance(x, dict) and "TERMINATE" == str(x.get("content", ""))[-9:].upper()
    )  # 终止条件，当收到的消息内容是"TERMINATE"时，结束聊天


# llm配置1
llm_config = {
    "model": "glm-4-0520",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "timeout": 60,
    "temperature": 0.8,
    "seed": 1234,
    "price": [0.01, 0.01],
}

# llm配置2
llm_config2 = {
    "model": "glm-4-plus",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "timeout": 60,
    "temperature": 0.8,
    "seed": 1234,
    "price": [0.01, 0.01],
}


judicial_decision = UserProxyAgent(
    name="裁判",
    is_termination_msg=termination_msg,
    human_input_mode="NEVER",
    code_execution_config=False,  # we don't want to execute code in this case.
    llm_config=llm_config2,
    system_message="你是裁判，你来判定哪方的发言更具有说服力，并且辩论赛的开始和结束都由你发起",
)

# 定义一个正方辩手
positive_man = UserProxyAgent(
    name="正方",
    is_termination_msg=termination_msg,
    human_input_mode="NEVER",
```

```
    code_execution_config=False,  # we don't want to execute code in this case.
    llm_config=llm_config2,
    system_message="你是正方，你认为电子游戏的兴起对青少年是有利的，你会搜索一些证据来证明你的观点",
)

# 定义一个反方辩手
negative_man = UserProxyAgent(
    name="反方",
    is_termination_msg=termination_msg,
    human_input_mode="NEVER",
    code_execution_config=False,  # we don't want to execute code in this case.
    llm_config=llm_config,
    system_message="你是反方，你认为电子游戏的兴起对青少年是不利的，你会搜索一些证据来证明你的观点",
)

# 用函数决定下一个发言者是谁
def custom_speaker_selection_func(last_speaker, groupchat: autogen.GroupChat):
    """Define a customized speaker selection function.
    A recommended way is to define a transition for each speaker in the groupchat.

    Returns:
        Return an `Agent` class or a string from ['auto', 'manual', 'random', 'round_robin'] to select a default method to use.
    """
    messages = groupchat.messages

    if len(messages) == groupchat.max_round - 1:  # 最后一轮由裁判结束任务
        return judicial_decision

    if last_speaker is positive_man:  # 正方发言后反方发言
        return negative_man

    elif last_speaker is negative_man:  # 反方发言后正方发言
        # Always let the user to speak after the planner
        return positive_man

    else:
        return "random"  # 随机选择一个说话者


# 定义一个聊天群组
groupchat = autogen.GroupChat(
    agents=[judicial_decision, positive_man, negative_man],
    messages=[],
    max_round=5,
    speaker_selection_method=custom_speaker_selection_func,
    allow_repeat_speaker=False,
)
```

```
# 定义一个聊天管理员
manager = autogen.GroupChatManager(
    groupchat=groupchat,  # 群聊群组
    llm_config=llm_config,  # 群管理员也可以配置LLM
)


def _reset_agents():
    judicial_decision.reset()
    positive_man.reset()
    negative_man.reset()


def chat():
    _reset_agents()
    chat_his = judicial_decision.initiate_chat(
        manager,
        message="辩论赛现在开始，辩题为：电子游戏的兴起是否有利于青少年的成长。双方每轮发言字数不能多于200。",
    )
    print("======================智能体处理已结束====================")
    print("\n")
    for i, v in enumerate(chat_his.chat_history):
        print(f"#{i+1} - {v.get('name')}>>>>")
        print("\n")
        print(str(v.get("content")).strip())
        print("\n" * 2)


if __name__ == "__main__":
    # 开始聊天
    chat()
```

autoGen结合Wikipedia搜索

```python
from typing import Union
from autogen import ConversableAgent, UserProxyAgent, register_function,
AssistantAgent
import wikipediaapi

GLM_OPENAI_API_KEY = "104328b036a52f886e055a58946ddc1c.tPxcvWdqRiorALOu"

llm_config = {
    "model": "glm-4-plus",
    "api_key": GLM_OPENAI_API_KEY,
    "base_url": "https://open.bigmodel.cn/api/paas/v4",
    "cache_seed": 42,  # 缓存种子
    "temperature": 0,
    "timeout": 120,
    "price": [0.01, 0.01],
}

# 初始化Wikipedia API
wiki_wiki = wikipediaapi.Wikipedia(user_agent="AutoGen")


# 查询维基百科
def query_wikipedia(query: str) -> Union[str, None]:
    page = wiki_wiki.page(query)
    if page.exists():
        return page.summary
    else:
        return f"没有找到关于'{query}'的维基百科页面。"


def is_termination(msg):
    print("is_termination参数:", msg)
    # 如果msg["content"]是空，则返回False
    if msg["content"] is None:
        return False
    # 如果msg["content"]是字符串，则返回"TERMINATE" in msg["content"]
    return "TERMINATE" in msg["content"]


user_proxy = UserProxyAgent(
    name="人类代理",
    # is_termination_msg=is_termination,
    human_input_mode="NEVER",
    code_execution_config=False,  # we don't want to execute code in this case.
    default_auto_reply="Reply `TERMINATE` if the task is done.",
    description="人类代理,可以向其他AI提问",
)

assistant = AssistantAgent(
    name="维基百科助手",
```

```
    system_message="你是一个维基百科的助手,你会帮助用户查询百科知识",
    llm_config=llm_config,
    human_input_mode="NEVER",
)

# 注册本地函数
register_function(
    query_wikipedia,
    caller=assistant,  # 函数的调用者
    executor=user_proxy,  # 函数的执行者
    name="query_func",  # 函数名，如果不指定，则默认为函数名
    description="在维基百科上查找知识",  # 函数的描述，决定函数是否被调用
)

user_proxy.initiate_chat(assistant, message="明朝是什么时候建立以及结束的?")
```

autoGen似乎不支持 像 crewAI那样直接设置远程工具，它只能注册本地工具，用 Wikipedia的python库来实现互联网查询。

接下来一篇将会用crewAI来实现类似案例。